

# CDMS Quick Reference

Quick reference for the CDMS module

## Files, (f)

```
f=cdms.open('myfile') # open the file 'myfile'
```

## Querying

### Global attributes

```
f.showglobal() # print the file's defined global attributes  
attdic=f.attributes # return a dictionary of the file attributes  
attlist=f.attributes.keys() # return the list of the file attributes  
nm=f.name # returns the value of an attribute (i.e. 'name'):   
f.myattribute='my attribute value' # to set an attribute to a new value:
```

### Dimensions

```
dims=f.listdimension() # returns a list of the dimensions in the file  
dims=f.listdimension('myvar')# returns a list of the dims of variable 'myvar'
```

### Variables

```
f.showvariable() # print the list of all variables in the file  
vardic=f.variables # returns a dictionary of the available variables  
varlist=f.variables.keys() # returns a list of the available variables
```

#### Variables in file: "File Variables"

```
dims=f.listdimension('myvar') # returns a list of the dimensions for 'myvar'  
f.showdimension('myvar') # same  
f.showattribute('myvar') # prints the attributes of 'myvar'  
f.showall("myvar") # prints details of the attr.s and dims of 'myvar'
```

## "Transient Variables" (=TV)

### Retrieving

```
tv=f('myvar') # gets 'myvar' from cdms file f
```

tv is now a TV containing the variable 'myvar' from the file 'myfile'

Let's say 'myvar' is 3D: time/latitude/longitude, if we wish to retrieve the longitude from -180 to 180:

```
s=f('myvar',longitude=(-180,180))
```

Note that by default the retrieval edges are closed/open, i.e the second value is not included in the retrieval procedure, if we wish to retrieve 180 then we would pass:

```
s=f('myvar',longitude=(-180,180,'cc'))
```

i.e.: closed/closed

Also cdms knows that the axis is circular, therefore even if the data are stored from 0 to 360, the extraction will be done correctly

Finally the procedure is the same for all dimension and can be mixed, for example the following are equivalent and retrieve the longitude from -180 to 180 (not included), the latitudes from -20 to 20 (included) and all the times:

```
s=f('myvar',longitude=(-180,180),latitude=(-20,20,'cc'))  
s=f('myvar',latitude=(-20,20,'cc'),longitude=(-180,180))
```

If you don't know the dimension name but know in which order they are stored you can do:

```
s=f('myvar',(:, -20,20,'cc'), (-180,180))
```

Note that ':' indicates that you want all the values of the first dimension.

Alternatively you can specify to retrieve a dimension by index, using the "Slice" function for example to get only the first 12 time step on the previous example you could do:

```
s=f('myvar',slice(0,12),(-20,20,'cc'),(-180,180))
```

Finally the time dimension accepts cdtime objects as arguments example to retrieve all the data in the year 1980 we would do:

```
import cdftime  
t1=cdftime.comptime(1980) # or t1=cdftime.reltime(12,'months since 1979')  
t2=cdftime.comptime(1981) # or t2=cdftime.reltime(24,'months since 1979')  
s=f('myvar',time=(t1,t2)) # remember, by default bounds are: closed/open ('co')
```

Now note that a sub selection of any TV is doable using the same syntax as for a file i.e.:

```
s2=s(latitude=(-20,20,'cc')) # get all the latitude in the range -20,20
```

## Querying

Querying a TV is similar to querying a file

```
attdic=s.attributes # returns a dictionary  
attlist=s.listattributes() # return a list of the variable attribute  
attlist=s.attributes.keys() # same  
dimnames=s.getAxisIds() # returns al ist of the dimensions names  
sh=s.shape # returns a tuple with the length of each dim.
```

```
s.info() # prints a description of the slab: attr. & dim.
```

## Dimension=Axis

### Retrieving

```
ax=s.getAxis(0) # returns the axes for 0th dimension
itim=s.getAxisIndex('time') # returns the index of the
dimension axlist=s.getAxisList() # returns a list containing all the axes
'time' tim=s.getTime() # returns the time axis
lev=s.getLevel() # return the level axis
lat=s.getLatitude() # return the latitude axis
lon=s.getLongitude() # return the longitude axis
```

### Querying

```
id=ax.id # returns the name of the axis
val=ax[:] # return a list of the axes values
att=ax.attributes # returns the attributes dictionnary
bounds=ax.getBounds() # returns a list of the 2 boundary of each coordinate
ax.isTime() # returns 1 if the axis represent time, 0 if not
ax.isLevel() # returns 1 if the axis represent level, 0 if not
ax.isLatitude() # returns 1 if the axis represent latitude, 0 if not
ax.isLongitude() # returns 1 if the axis represent longitude, 0 if not
ax.isCircular() # returns 1 if the axis is circular
ax.modulo # returns the value of the modulo (for circular axis)
```

## Altering

```
ax[:]=newvalues #to change the values of an axis (Ax):
ax.id='my new name' # Changes the axes name/id
ax.units='my new units' # changes the units
ax.myspecialattribute='my special attribute value' # set the attribute
ax.designateTime() # sets the axis as designating time
ax.designateLevel() # sets the axis as designating levels
ax.designateLatitude() # sets the axis as designating latitude
ax.designateLongitude() # sets the axis as designating longitude
ax.designateCircular(value) # sets the axis as circular, modulo "value"
```

### Writing data to a file

Let's suppose we have a TV having the dimensions: time,latitude,longitude, with the dimension set unproperly, and we want to write this TV (let's call it tv) to a file. Let's assume the shape is 12,64,128 and the grid is gaussian T42 (from -180, to 180 and south to north), and represent the 12 months of 2000

## 1–Preparing the dimensions/axis

```
time=cdms.createAxis(range(12)) # Create the "raw" axis

time.id='time'
# set the name

time.units='months
since 2000'

time.designateTime()
# specifiy the axis as "time" (independantly from the name)

lons=MV.arange(128,typecode=MV.Float)*2.8125-180. # create the values for the longitudes
lon=cdms.createAxis(lons)
lon.id='longitude'
lon.units='degrees_east'
lon.designateLongitude()
lat=cdms.createGaussianAxis(64) # creates a gaussian axis with 64 latitudes
lat.units='degrees_north' # Now we need to flip the values, because the default generate from N to S
bnd=lat.getBounds() [::-1] # retrieve and flip the bounds
lat[::-1]=lat[1:-1] # flip the latitude values
lat.setBounds(bnd) # reset the bounds
```

## 2– Setting the axes into the TV (tv)

```
tv.setAxis(0,time)
tv.setAxis(1,lat)
tv.setAxis(2,lon)

# Alternatively

tv.setAxisList([time,lat,lon])
```

## 3– Writing to the file

```
f=cdms.open('myfile.nc','w') # to create a new dataset
f=cdms.open('myfile.nc','r+') # to open an existing file
f.write(tv)
# or if we're not sure that we set the axes on the tv:
f.write(tv,axes=(tim,lat,lon))
f.close() # if not the program might end before data are actually written
```

## 4– Unlimited dimension

by default time is set as unlimited and can be extended for example:

```
for i in range(len(time)):
    t=tv[i]
    f=cdms.open('myfile.nc','r+')
    f.write(t)
    f.close()
```

would write all the time one after the other, even though it doesn't make sense to open and close the file everytime !

## 5– Full form of the write function

```
fv = write(var, attributes=None, axes=None, extbounds=None, id=None, extend=None, fill_value=None)
```

\*var is a variable or array.  
\*attributes: is the attribute dictionary for the variable. Use this to specify attributes if var  
\*axes is the list of file axes comprising the domain of the variable. Use this to set the axes if  
\*extbounds is the extended dimension bounds. Default: var.getAxis(0).getBounds()  
\*id is the variable name in the file. Default is var.id.  
\*extend=1 causes the first dimension to be 'extensible': iteratively writeable.  
    The default is None, in which case the first dimension is extensible if it is time.  
    Set to 0 or None to turn off this behaviour.  
\*fill\_value is the missing value flag.  
\*index is the extended dimension index to write to. The default index is determined by lookup rel